

Resumen Para el Final

Arquitectura

Es el conjunto de recursos accesibles al programador. Es lo que se mantiene p/ lograr compatibilidad

- Registros
- Set de Instr
- Estructuras de Mem (Dep de reg)
- Diseño lógico

Microarquitectura

Es la implementación física de la arquitectura y puede modificarse ~~sin~~ dejando a la vez intacta la arquitectura

Ej: IA-32 va del 80386 → i7 (con + de 25 millones)
 - Diseño del circuito, alimentación, refrigeración

Tipos de ISA

Clases: Registros → prop. genl
 ↓
 dedicada

Registros-Mem / Load Store

Diseño de Memoria: Almacenamiento obligatorio / Address Bytes

Modos de Direccionamiento: 3 + tipos de especificar los operandos

Tipos y rangos de operandos: int / float word / double

Operaciones: Simple (RISC) / Muchas (CISC)

Conte de ctrl de flujo: Salto condicionales, call.

Long del código: Instr de tam fijo / variable

Microorg = Organización + Hardware

Organización: Serán los detalles de la implementación de la ISA

- Organización e interconex de los Man - Diseño de los bloques de la CPU - Impl de Paralelismo a nivel de Instrucciones / Píctos

∃ Proc de == ISA y org ≠: Ex: AMD FX vs i7 (x86?)

Hardware: Son los detalles ^{implementación} del diseño lógico y tech de fabricación, electrónico.

∃ proc == org y ≠ Hardware, el Pentium 4 y 4M P!
(El Pentium 4M mayor optimización p/d como electrónico)

"Esto permite ventajas de no tener que recompilar un binario si se cambia el procesador, & tener == Org"

"Ej la org x86 se implementa en el Pentium 4 con NetBurst y en el AMDathlon con uno de AMD"

cada etapa se
le llama stage

1 instr x ciclo de
clock

Máxima utilidad de unidades,
línea de montaje

HOJA N°

FECHA

Instruction Level Parallelism

Pipeline: org que permite crear el efecto de
superponer a el tiempo la exec de varias instr a la vez.
Se logra haciendo que los bloques funcionales trabajen
en paralelo c/u en una instr diferente. (Stages)

"Sería como una línea de Montaje"

Se busca que el stage sea lo + peq (7 ciclo de clock)

No se reduce el T de exec de c/ instrucción individual
(^{x overhead} incrementa), lo que si es que se aumenta el
Throughput

El TPI = tiempo de instr en pipeline / # stages

	F	D	E	R
1	A			
2	B	A		
3	C	B	A	
4	D	C	B	A
5		D	C	B
6			D	C
7				D

→ Estabilidad del pipeline

++ Throughput

++ overhead

P! Aparece en la instr 4 y 5

En teoría alargar el pipeline mejora la performance
(pues ++ frequency), pero se incrementan los penalties

La idea es que hagamos la mayor cant de cosas en paralelo
Natura tiene 31 stages (stage), Core i7 14

NOTA

Hazards: Structural, Data & Control

Conllevan a un Pipeline Stall -- perf

Structural:

- Causas
- 1) Una etapa no suficientemente atom lleva + de un ciclo de clock
 - 2) Dos instrucciones en 1 etapa requieren la misma unidad funcional

Penalty: Una de las dos instrucciones no se puede completar $\Rightarrow +1$ ciclo de clock

$$CPI \leftarrow CPI + 1$$

Se incrementa
#Comunicaciones
-1

Ej: Si la unidad de fetch de instrucciones atiza también se utiliza para hacer el operad fetch

	OpCode	F	DEC	Operand	F	Execute	Retire
1	A						
2	B		A				
3	C		B	A			
4	C		-	B	A		
5	D		C	-	B	A	
			D	C	-	B	
				D	C	-	
					D	C	
						D	C

(- son NOP, o "bubblers")

Possible solutions

A) para acceso a memoria:

- 1) Duplicar el Cache en 2 (Uno p Datos y otro Instr)
- 2) Bufferizar las instrucciones como FIFO
- 3) Buses + anchos (+ cost)

B) Aumentar la profundidad del pipeline

se produce en 2: - true RAW

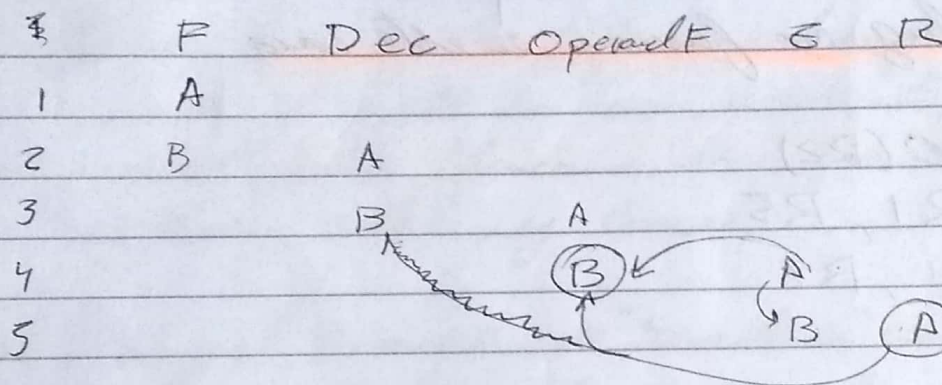
- name dep antedep WAR
output dep WAW

HOJA N°

FECHA

Data Hazard: Se produce cuando por efecto del pipeline se requiere de un dato antes de que este este disponible por la unidad del programa

Ej: $\text{add } R_1, R_2, R_3, A$
 $\text{sub } R_4, R_1, R_5, B$



Se prepara $CPI = CPI + \frac{n}{m}$

→ Distancia entre etapas que requieren el mismo dato

Possible Solutions

1) Forwarding: se elige el resultado directamente de la unidad de ejecución y se lo envía a la etapa en el mismo ciclo de clock en que se hacen los op destino.

(Se aplica solo a las etapas que quedaban a stall)

2) No Soluciona pero mejora el prob

	F	D	OF	E	R
1	A	A			
2	B	A			
3		B	A		
4			-	A	
5		B			A
			B		
				B	

(Alcudo sig al escritura
al op = fetch) ✓

RAW: Se lee un dato
antes de que este haya sido
generado

WAR: Se escribe un dato
antes de que haya sido leído
y hace que se lea un dato
incorrecto

WAW: Se inserta el
orden de escrituras

→ pueden por OoE

Ⓐ Sucede + bus xq el W tarda

El forwarding no funciona en este caso

ld R1, 0(R2)
sub R4, R1, R5
and R6, R1, R7
⋮

Nada puede adelantarse hasta que no este el load

Control Hazards: "El código no es lineal" 7 jumps

Los Branches son responsables de frenar (u
no se se produce) todo el flujo del pipeline
hasta que se recupera el jump

El Branch penalty == # stages (deep → + penalty)

	F	D	OF	E	R
1	SW				
2		S			
3			J		
4				S	
5					J
6					

Tengo que
Flushear

Nota A/B

Se puede determinar en la fase

Expresión \rightarrow salto condicional

Op F \rightarrow salto ^{o llamada} incondicional c/ dir indirecto

Decode \rightarrow " " " " " " directo

Unidades de Predicción de Saltos

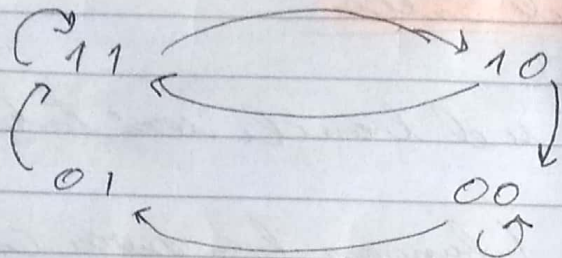
Se busca predecir si el branch será taken o no

-) Predict non-taken (funciona bien para casos donde el salto es hacia adelante)
-) predict taken (funciona bien cuando el salto es hacia atrás, ej. loops)
-) Delayed Branch: se "reordena" una instrucción mientras el branch se ejecuta, Delay Slot
Ej: si no son dependientes se puede ejecutar una anterior.
Si no, se puede ejecutar una del target o el fall through, pero habría que checkearlos en el caso de misprediction (~~Predict Exception~~)
-) Predicated Execution: convert Branches to P!
conditional Moves
-) Fine-Grained Multithreading: "do something else"
(ST perf --)
-) Loop Unrolling (puede ser en el cache)
Es necesario que el loop sea paralelizable

Predicción de Saltos Dinámica

Branch Prediction Buffer:

Tabla indexada con la dir del branch y con (o dos bits) para taken/not-taken



90% accuracy

arregla el problema de TTTTNTTTT \rightarrow 2 pred mal
(y find max) 1 pred mal

Esta prueba va a el IFetch, pero también podemos agregarle bits al cache en si

- Se vio que no existe mejora significativa a alla de 2 bits y 4K items

Branch Target Buffer:

Cache de instrucciones de salto que contiene p/c cuando la dirección target resultó

Se actualiza con el PC

Se assume taken (o no está)

• taken \rightarrow se agrega al BTB

• not taken \rightarrow no se reemplaza

en esta

y esta bien \rightarrow OK

mal \rightarrow se reemplaza por el nuevo target

Two-level Predicta

R1: A branch outcome can be correlated with other branches outcomes

• Global Branch Correlation:

: Hay que guardar la historia en un Global History Register y usarlo p/ indexar una tabla que tiene el target p/ correlar de GHR

Se llama PATTERN HISTORY TABLE (2 bit sat counters)

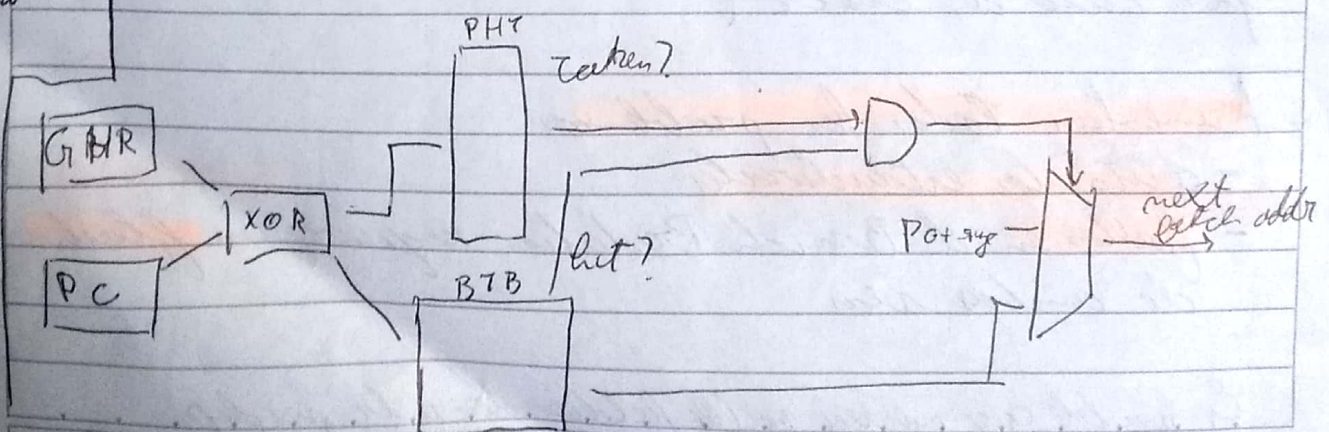
Ej. Pentium Pro 4bit GHR con multiple PHT

XOR: Se puede añadir contexto a la PHT si la indexa con un hash del address y el GHR

Realization: el outcome de una branch se puede correlacionar con el " de ella misma (además de su última vez)

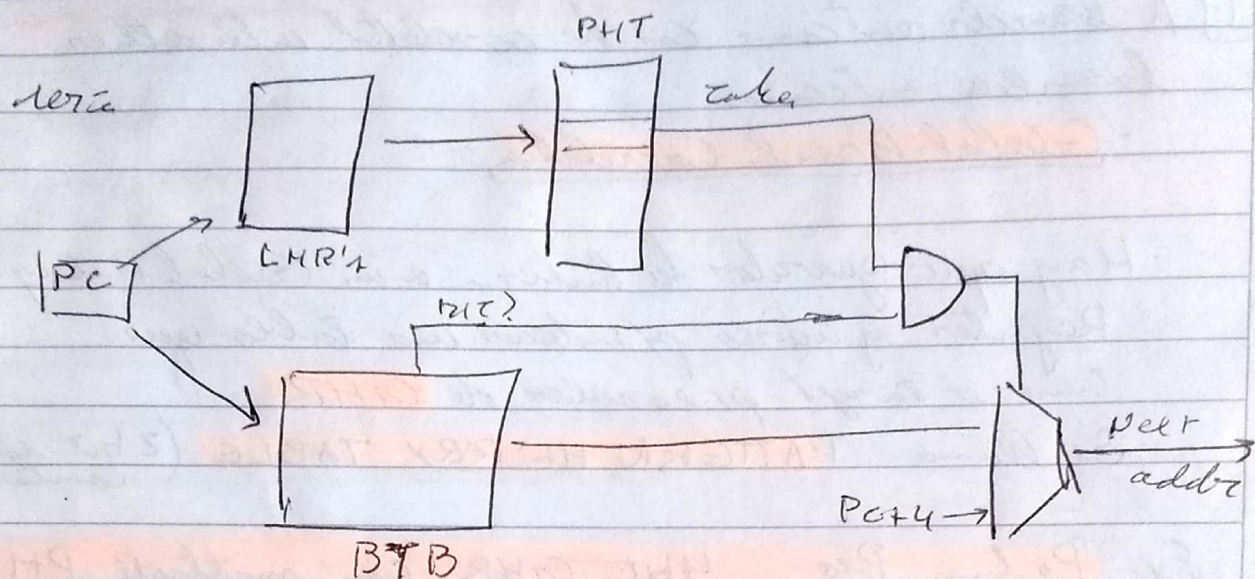
ver dopo
de summer
taken or
no enter

• Tener un HR por branch (es caro!)
(local history branch pred)



NOTA

Si es LHR's entonces tenemos una tabla en vez de un solo registro



Existen combinaciones en forma de Hybrid Branch Predictors y tournament predictors (Hay otros tipos mas como "perceptrons")

Todo: Multi-path execution / Indirect Branch pred (Over 10)

Superscalars: (Pentium 2 vías)

Se intenta ejecutar más de una instrucción por ciclo de clock.

Aumentan todos los problemas:

- Altimos estructurales
- fallas en el Branch Prediction requieren un flush de ambas vías

Es posible que no sea posible fetchear dos instr. por dep.

Notas \Rightarrow Hyper Threading mentions a los componentes ocultos

Scheduling Diagrams: Ejercicios fuera de Orden

div F_0, F_2, F_4 } dependen
add F_{10}, F_0, F_8 }

sub F_{12}, F_8, F_{14}

→ no hay matrices p/p entre las puercas ejecutables

IN ORDER

div	F	D	O	E	E	E	E	R	W				
add		F	D	O	stall			E	R	W			
sub			F	D	O	stall			E	R	W		
mul				F	D	stall				O	E	R	W

Out of Order

div	F	D	O	E	E	E	E	R	W	
add		F	D	O	wait			E	R	W
sub			F	D	O	E	R	wait		W
mul				F	D	O	E	R	wait	W

div F_0, F_2, F_4
add F_6, F_0, F_8
sub F_8, F_{10}, F_{14}
mul F_6, F_{10}, F_8

- Races
- ① Sub escribe en operando (WAR) antes
 - ② mul escribe en output de add (WAW)

Excepciones Imprecisas y Branch Mispredict

Sea cual sea la solución implementada, es necesario que esta no dificulte la tarea de debuggear un programa, es decir que sea posible en cualquier momento ver el estado de los registros con el código ejecutado hasta cierto punto

Puede suceder que una instrucción que se haya ejecutado no debiera haber sido o viceversa

- un

Reorder Buffer

Scoreboarding: primer intento de COPE

+ Structural Hazard
+ (No Forwarding)

Se utiliza para prevenir ~~errores~~ errores de falsas dependencias (WAW y WAR). Pero no los resuelve

El Scoreboard generará un stall deteniendo la instrucción de ser necesario (también detecta RAW)

Se divide desde el Issue y Read Operands, quedando

- 1) Fetch

- 2) Issue: se decodifica y detectan dependencias (Chequeando el scoreboard). Se frena la ejecución (stall) en caso de detectar un conflicto WAW

- 3) Read Operands: la instrucción espera hasta que estén listos sus operandos (evitando así RAW)

- 4) Execute: se ejecuta y notifica al scoreboard.

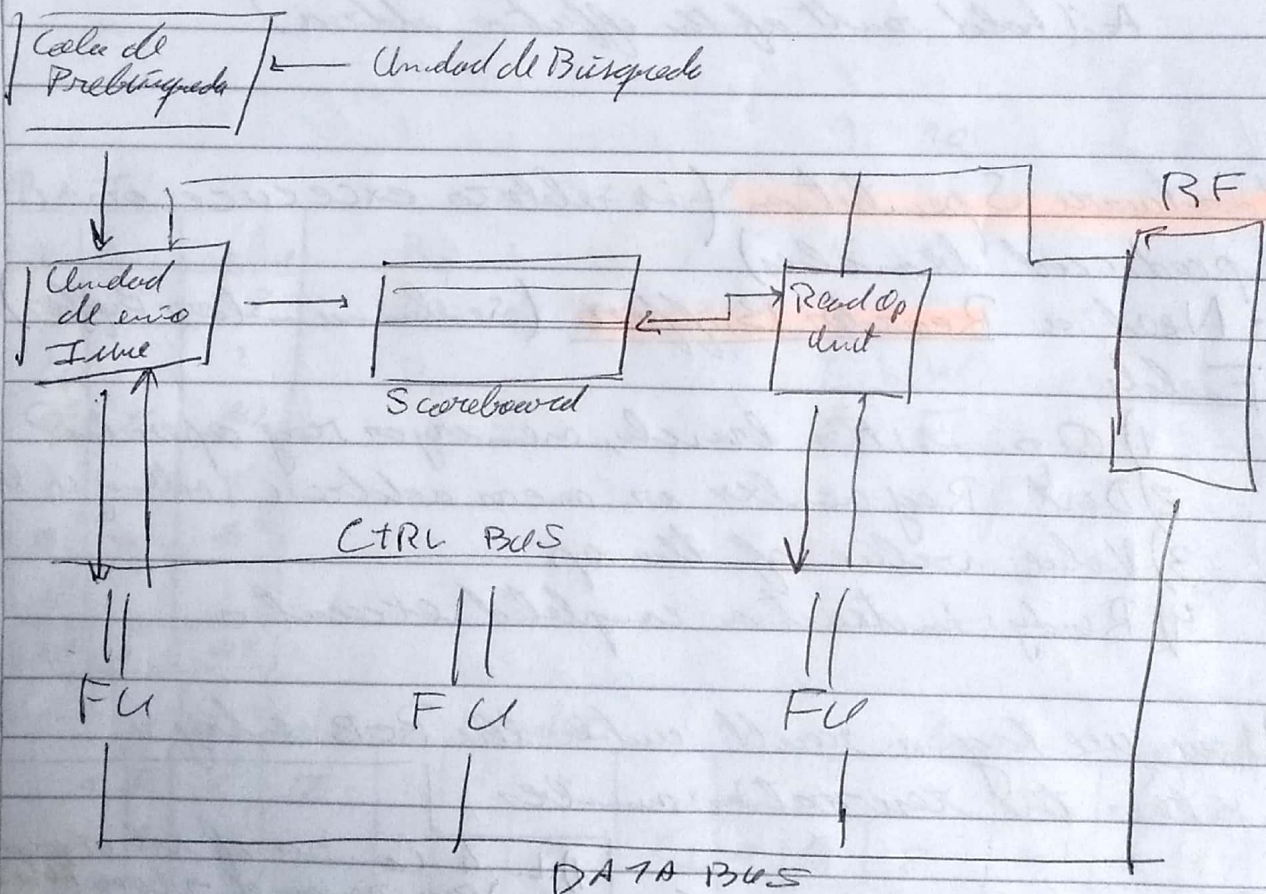
- 4) Write Back: En esta instrucción se escribe al registro destino, pero se espera a que instrucciones previas hayan leído el registro (entonces WAR)

Problema adicional: control Hazard

Este método realiza un stall en el issue stage de no haber UF dependencias.

Tramitamos resolver este stall y WAR y WAW con Register Renaming.

ScoreBoard



Algunos problemas (dopo fixifi)

- ++ Structural Hazards, pues controla buses al limitador p paralelizar transf entre scoreboard y registros
- Operando se lee de los registros y no se aprovecha Forwarding

NOTA

Tomasulo: Register Renaming

Reservation Station

Op Q_i Q_i V_j V_i A Busy

Register File

Q_i

of Reservation Station that contains op to be stored here

Load/Store Buffer

A: (hold result of the effective address)

Hardware Speculation (is able to execute on predicted branches)

- Need a Reorder Buffer (se llama Store Buffer)

Fields

- 1) Op: is it a branch, memory or reg operation?
- 2) Dest: Reg number or mem address (nothing if branch)
- 3) Value: value of the op
- 4) Ready: instruction completed execution

Now we tag a result with the ROB entry rather the reservation number

⊕ Preempt Interruptions ✓

- 1) Link Camion of val of production
• Reg Renaming → lower value deq
- 2) Guardar los enter hasta ready
• RS → permite seguir con el op
- 3) Ir chequeando que enter este listo
• Guardar comandos que se
→ permite cancelar
- 4) Cuando todos listos, dependencias
• wake up → permite OoO

GROOT ——— am

① Check RS spot

② Allocate val / tag on RAT

③ Remove output (set Reg)

~~for each GROOT = PDU command~~

Ej: MUL R₁, R₂ → R₃

RAW ADD R₃, R₄ → R₅

ADD R₂, R₆ → R₇

C ADD R₈, R₉ → R₁₀

y MUL R₇, R₁₀ → R₁₁

d ADD R₅, R₁₁ → R₅

1 2 3 4 5 6 7 8
F D E₁ E₂ E₃ E₄ E₅ E₆

F D — — — — —

F D E₁ E₂ E₃ E₄

F D E₁ E₂ E₃

F D — —

F D —

RAT

R₁

R₂

R₃

R₄

R₅

R₆

R₇

R₈

V	tag	value
1		1
1		2
1	X	2
1		4
1		6
1		6
1	b	8
1		8

R₉

R₁₀

R₁₁

N tag val

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

N

1

1

1

Otro ex

odd R_1, R_3	F D E E E E E E ... ERW
even R_6, R_7, R_8	F D - - - - EG-ERW
odd R_{11}, R_{12}, R_6	F D - - - ER
even R_3, R_3, R_2	F D E R
odd R_7, R_8, R_8	F D E R
odd R_{10}, R_9, R_3	F D E R

Hay que notar que el Hardware Speculation hace que no solo se pueda un loop en la RS, si no también uno en el ROB (si no hay \rightarrow stall)

Se explican las 4 etapas con ROB

- 1) Envío: se encarga de chequear que E ilot en RF y ROB p/ meter fletcherada y actualizar sub c/info
- 2) Ejecución: Si faltan apt se monitorea el CDB aguardando el broadcast (chequea RAW), y ^{no ejecuta}
- 3) El Resultado se escribe en el ROB y se broadcastea a las RS que lo necesitan
- 4) Commit: Si se trató de un branch con pred incorrecta se flusha el ROB y hay que empezar de nuevo (desde la vez correcta)
Si no es un branch se escribe el resultado a donde corresponde (reg/mem)

Case: Practico: microarquitectura, P6 (1993)

3 cores engine: Duramax 5.3L (no es supercalor, y no tiene una ventosa de inyección)

Tambien implementa ODE y Hordenware Spec

- Controla los 3 cores. C/ el mundo exterior
- Busca en el L2 datos e instrucciones (4 accesos concurrentes)
- Controla el acceso al system bus mediante snooping

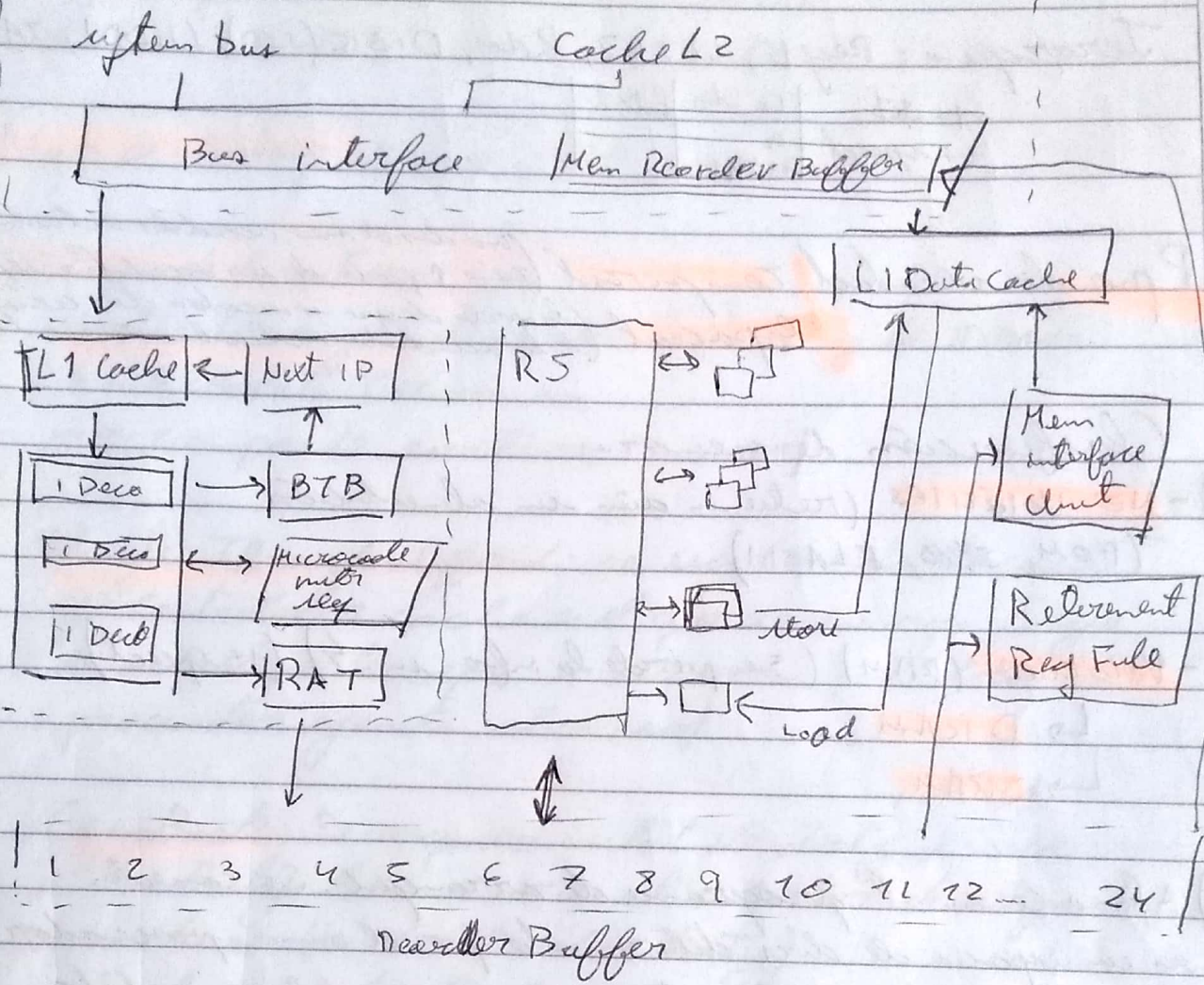
HOJA N°

4651

Cache de instr
L1 Cache
hace
16 bytes

El BTB
de el NextIP
Se desactiva
las instr
en uOps
que se
cancelan el
RAT
que hace
Reorderning

A la vez, el
allocator
de los
estados
cancela
de los instr
en muy largo
el microcode
instr
requieren
que los
uOps
cancelaron



- La reordenación S. verifica que la uOp tenga los datos req y envía a la unidad.
- Los saltos se marcan en el ROB con la dir predicha por el BTB para luego verificar si estuvo bien o no.
- El RRF se encarga de reordenar los operandos de la ROB en la IA (puede Reorder 3 uOps por clock).
- La interfaz de memoria aplica en el L1 Cache los datos a una una ep de memoria.

NOTA

El Sistema de Memoria

Hierarquía: Regl., L1, L2, RAM, DISK (SSD), (HDD), TAPE

++ \$	La idea	bitrate
++ speed	or	
	++ AMAT	

Punto de velocidad Temporal (para dar de men. accedidos actualmente) (tome + prueba de ser accedido en el futuro)
Espacial (la proba de que se acceda a dir. cercana) (o la accedida actualmente o alta)

Clasificación de memorias

- A) - **NO Volátiles** (retienen aún sin alimentación)
(ROM, ~~SSD~~, FLASH)
- B) - **Volátiles** (RAM) (Se pierde la info en \downarrow) (++ speed)
 ↳ **DRAM**
 ↳ **SRAM**

A) Almacenan el programa de arranque. Se conectan en un espacio de dir. determinado por el núm. procesador. Se busca la 1ª instrucción luego del encendido.

B) DRAM 1 bit = 1 transistor
Necesita regenerar la carga una vez que se lee (esto la ralentiza)

SRAM 1 bit = 6 transistores + cara + rápida (lee más despacio)

	DRAM	SRAM
Comun.	POCO	MUCHO
Capacidad	ALTA	BAJA
Costo	BAJO	ALTO
Tiempo	LENTO	RAPIDO

Notas

Soluciones para SRAM en la Cache:
definiciones de hit, miss y hit rate

Algos de desplazo: Round, FIFO, LRU, LFC

Políticas de Escritura:

Write-Through: se escribe la cache y la DRAM

++ Absoluta Coherencia

++ tiempo de escritura (= DRAM)

Write-Through-buffered: se escribe solo la cache
el controlador cache es el que se encarga luego
de actualizar la copia en memoria (mientras el
procesador ejecuta otra cosa)

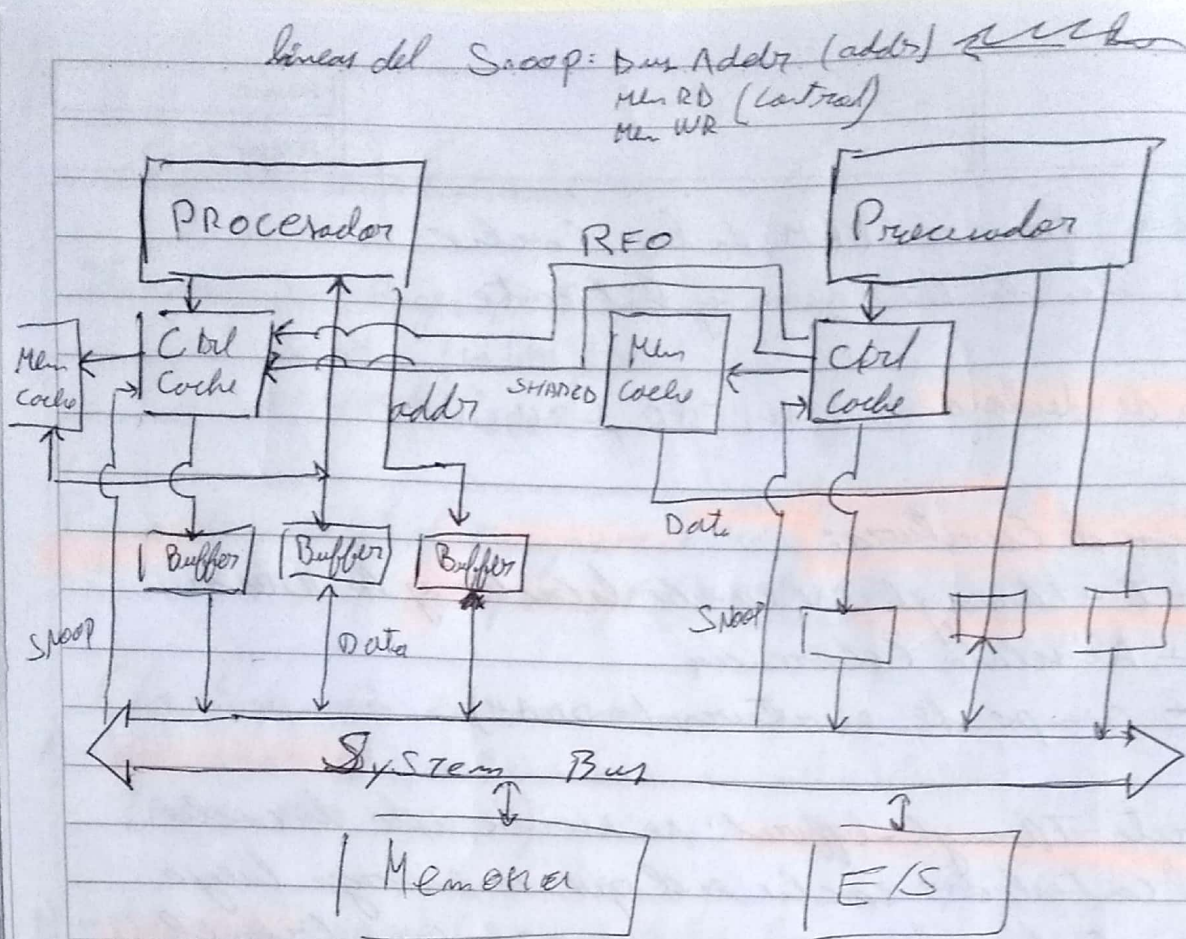
CopyBack: Se requiere un bit de dirty, Solo se
actualiza la memoria al intentar de ~~salir~~ ~~de la~~
línea (u el bit está dirty)

Si el procesador hace más mientras se está actualizando
el valor, se debe esperar hasta que el ctrl de cache
haya terminado la actualización

¿Se puede usar Copy-Back
EN SMP?

SI, se necesita coherencia

\$60 x
152
NOFA



- El ctrl de cache tiene un directorio interno en el cual guarda qué líneas (direcciones) tiene almacenadas la cache
 - Con el **Snoop Bus** espía / ojea de lectura y escritura p/ ver si la tiene (la dir.)
 - Si la tiene y detecta una escritura: la invalida
- "No es un bus entre los controladores, sino una conexión que CPU toma p/ espiar que sucede en la memoria"

Para que el Snoop Bus funcione con la partición + eficiente (copy-back), es necesario un protocolo de coherencia

Snoop {Addr, MEMRD, MEMWR}

Protocolo de Coherencia: MESI

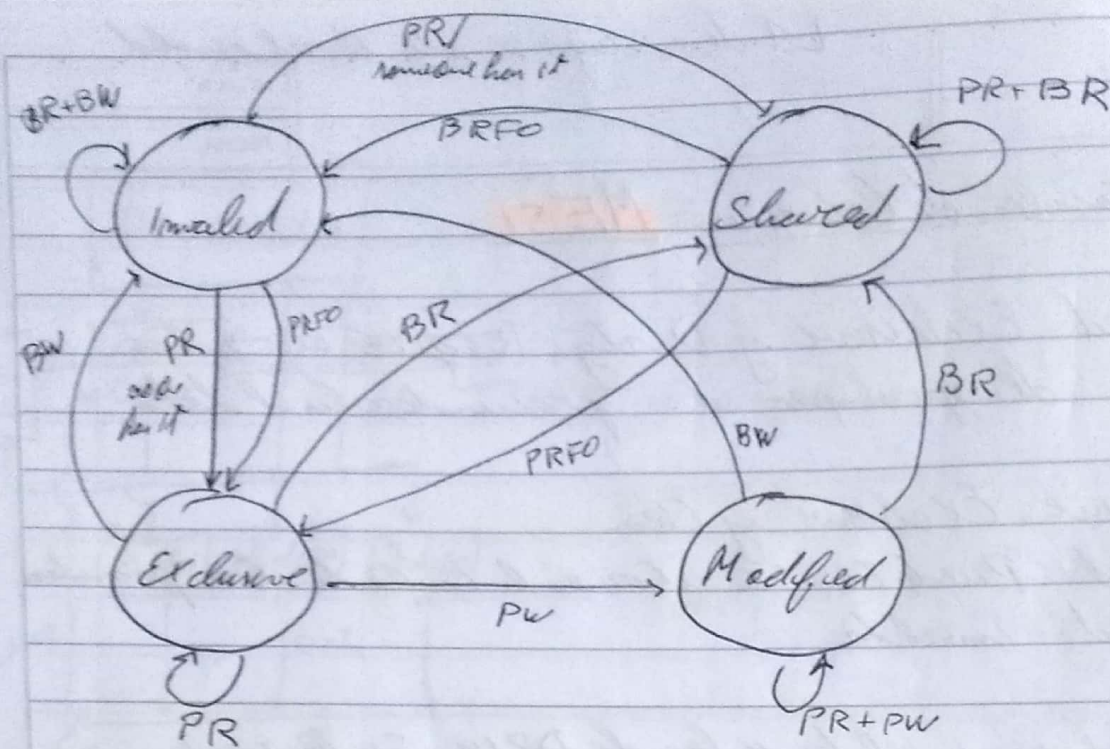
Ownership { Modified: Exclusive y Dirty. Requiere write antes de que el ~~pro~~ otro procesador lea el dato

Exclusive: Exclusive y Clean

Shared: Puede también estar en la cache de otros procesadores

Invalid: invalida.

- Las líneas invalidas se leen de DRAM. Si las tiene otro ctrl cache, este la puede proveer marcándola previamente como "SHARED" (o Exclusive)
- Por el línea Exclusive, se monitorea el SB por transacciones que la incluyan. De haber una, se pasa a Shared, se activa la línea Shared y hace un broadcast de la address (El que lo lee también la marca shared)
- Si hay que escribir una Shared se activa RFO, que hace que el resto la pase a invalid
- Si se detecta por el Snoop Bus una lectura de una Modified
 - 1) Activa RFO
 - 2) Escribe a la DRAM el valor modificado
 - 3) El lector copia ese dato a su cache ~~allí~~ (cuando aparece en el bus de datos)
 - 4) Ambos lo marcan Shared



P: "Estado ppal"
procesador

B: "otro procesador"

only

g) Write Back on M "Downgrade"

- 1) Salvo que linea I, todas las lecturas van desde el cache
- 2) Las que son I se leen en la Ram o de otro cache si la tiene (actuando linea shared). La pone en Shared o Exclusive.

Ej: si otro la tenía en Exclusive, la cambia a shared activa la señal y pone el valor en el bus. El que la lee también la marca shared.

- 3) Si la linea es Modified o Exclusive puede escribirse sin pasar por la DRAM.

4) Si se necesita escribir una shared: ^{o invalida} hay que invalidar el resto con 'RFO'.

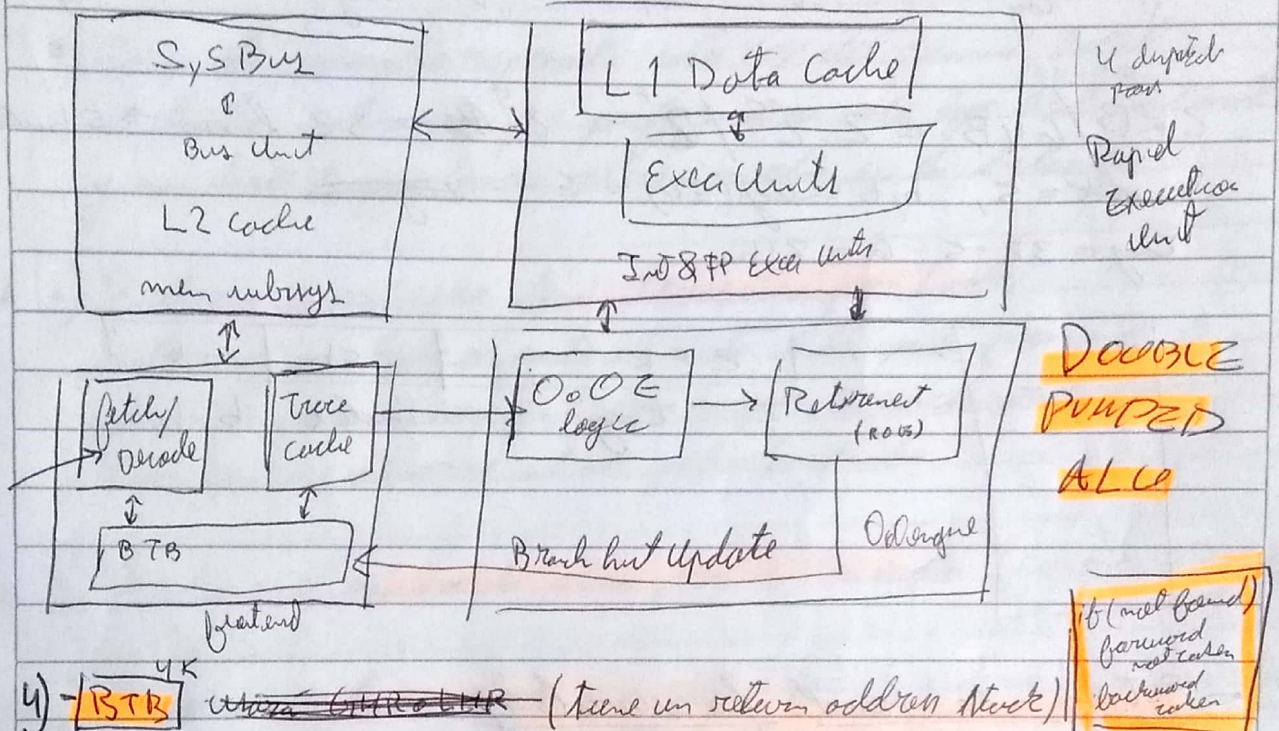
- 5) Si hay una lectura de una linea Modified: Se activa ^{Shared} RFO p/ avisar que el dato está inconsistentemente, se escribe en DRAM el dato (y el lector lo copia) y vuelve la linea a Shared.

P1 The Microarch of Pentium 4

1) - Describe la micro **Net Burst**

2) - Nuevo: **Execution Trace Cache** (Cache de uops (instr ya decod))

3) - **OoOE**, **ROB** (retire 3 uops per cycle)



4) - **BTB** ~~Branch Target Buffer~~ (tiene un return address stack)

5) - **higher clock** → deeper pipeline → 50% clock increase → 30% perf

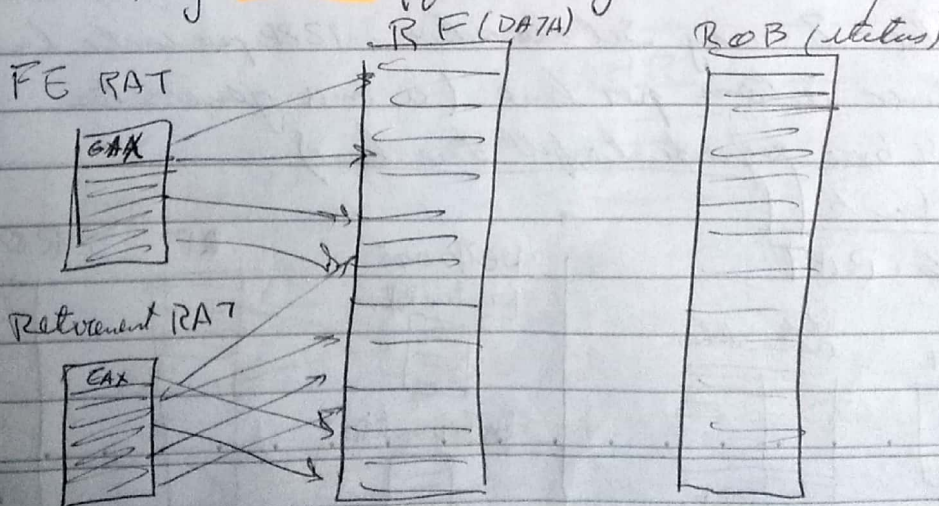
6) - **deep** (Netburst) > 2 deep (P6), clock (Netburst) > 1.6 clock (P6)

7) - La **misprediction** pipeline es mucho + larga

8) - Tiene **tan solo** con ROB

9) - 128 entry Register file

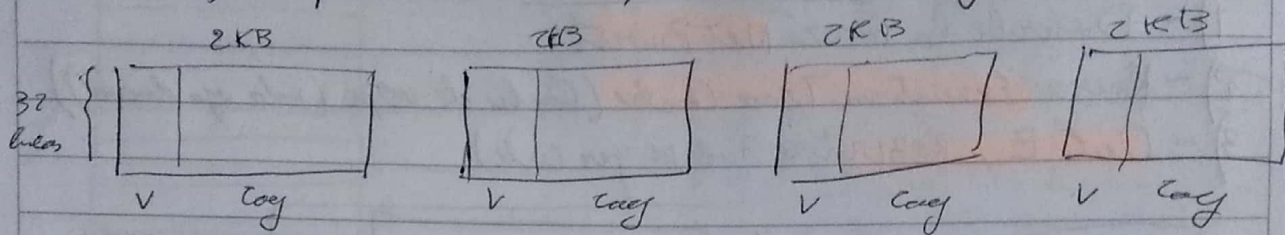
10) - **2 RAT** (frontend y retirement que apuntan al RF)



NOTA

miss data speculation (7)

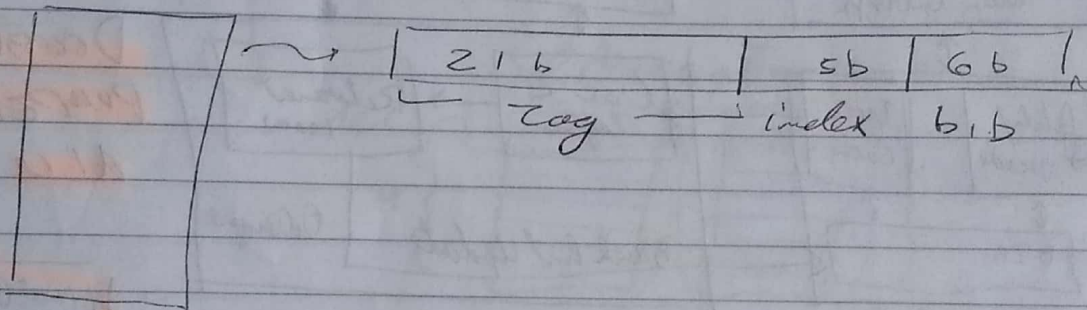
- Cache L1 Data 8KB, 4 way, Set Associative
64 bytes per cache line, write-through (to L2)



$$2KB/64B = 2 \cdot 2^{10} / 2^6 = 2^5 = 32 \text{ lines} \rightarrow 5 \text{ bits}$$

$$\text{index} = 5, \text{ b.i.b} = \log(64) = 6 \quad \text{tag} = 2$$

$$\text{tag} = 32 - 5 - 6 = 21b$$



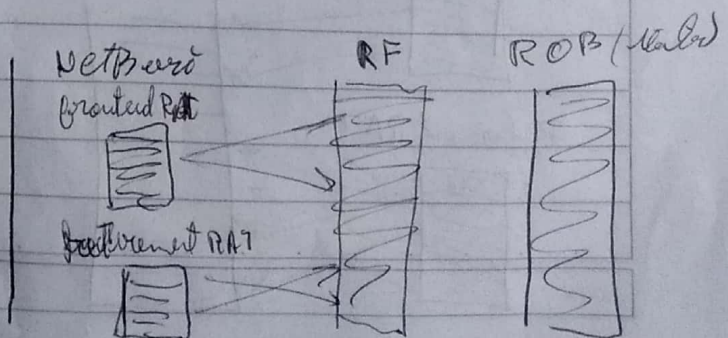
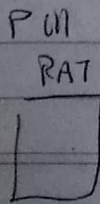
- Store-to-load forwarding

Pending Store Buffer: los loads pueden usar el resultado de los stores aunque no hayan commitados
24 entry store forwarding buffer
(tienen que tener $= 0 < \text{time to execution}$)

- L2 cache 8-way set associative 128B per cache line
~~Write back~~ Two sectors per line (a miss generates two 64 byte requests to fill the line)

write back

Dufg / P6: RAT



~~Unit 29 Cache Coherence~~**P3** The Intel Pentium M, Micro and Perf

Power dissipation & battery life

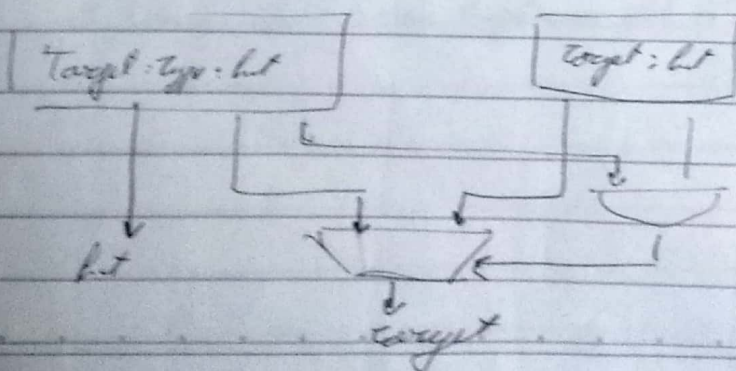
KW- TRADEOFF, BP, STACK Manager, microOp fusion, bin

- Como ~~el~~ ^{el} procesador representa solo 10% del consumo total se debía optimizar p/ performance solo bajo ^{THERMAL} heat constraint y no darle importancia al impacto en la batería

- A) Reducir el número de instrucciones por task
(el # es fijo desde el punto de vista de la arch pero no desde la microarch.) Ej: un mejor Branch predictor
haciendo que se ahorre al no pifarla tanto
 - B) Reducir el número de uOps por instrucción (fusión y stack pop)
Ej: Equi
 - C) Reducir # transistores por uOp (fusión y re-uso)
Se ahorra la POP de ESP
 - D) Reducir Energía por transistores unit
(Speedstep reduce el voltaje operacional en períodos de bajo uso)
(Capacitor Discharge of BOB full)
- A) Al BP del PM se agregan 2 predictores: loop detection & Indirect Branch predictor
IBP: funciona análogo al GHR pero con direcciones en vez de retidos

IP

GHR



B) μOps fusion: fewer μOps → less energy spend
Ej: store y loads se splittean en 2, pero fusionamos
ambos roles en la ejecución actual (antes y después
van juntos)

Stack engine: hace los esp. más efectivos en la etapa
de decode (optimizando push, pop, ret, call...)

C) Bus nuevos protocolos y circuitos

D) Identify idle logic & shut it down

Ej: cuando se llena el ROB, solo quedarse llenos con
nada, por lo cual se retira el pipeline y el Allocator
expone un clock

SpeedStep® (7000)

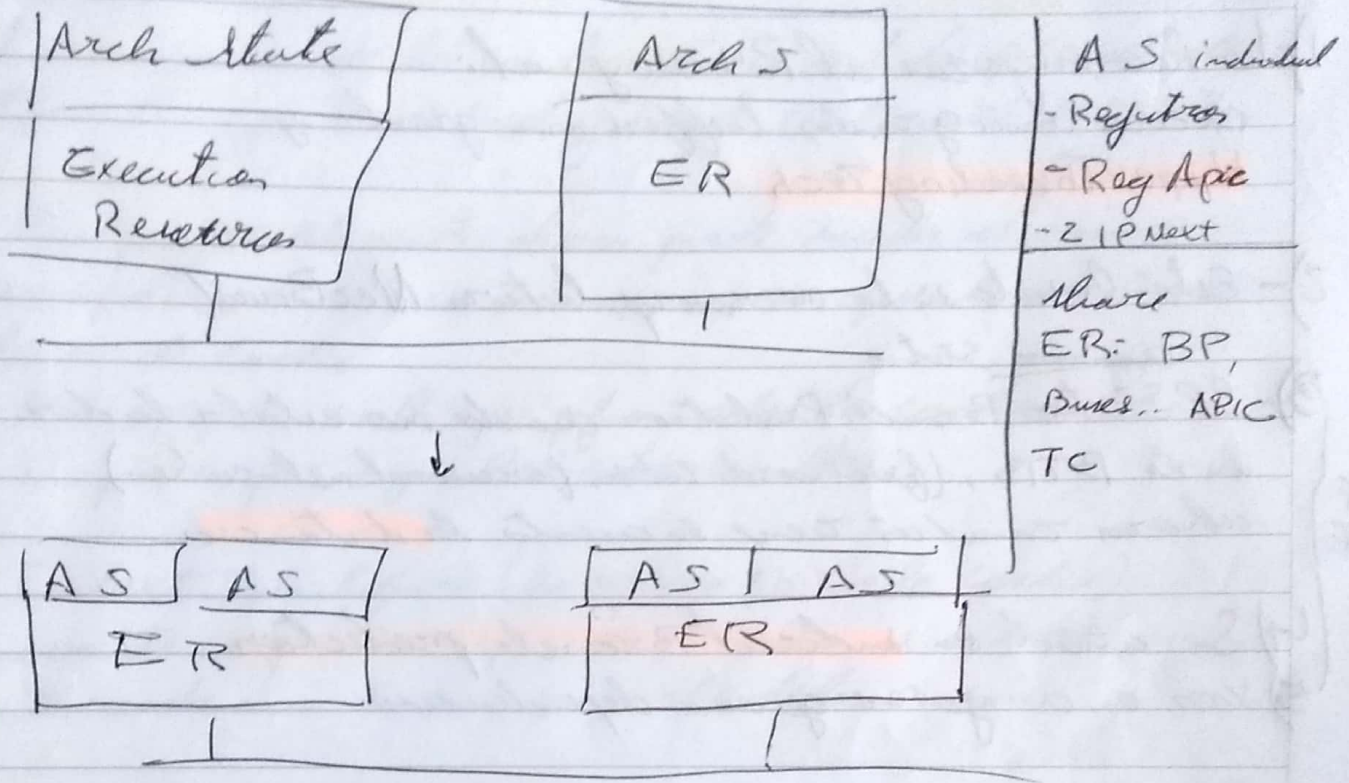
Las unidades comienzan su ejecución sub-clockadas
y aumentan la frecuencia a medida que
la operación lo requiere

P2

HyperThreading Tech Arch & Microarch

- 1) - Simultaneous Multithreading → HT tech (R)
- 2) - The arch state is duplicated (pero no los UE) GPR
CRX
APIC
RAT
- 3) 1) { La arquitectura se ve como 2 procesadores +
y puede scheduler threads así, pero realmente solo
hay 2 arch states "
motivación:
- 4) - Seguir aumentando el clock freq deja de ser eficiente por el pipeline y vuelve muy largo y los penalties a la vez más caros. A la vez añadir UE deja de ser eficiente por el paralelismo es limitado en el data flow.
Es decir el ratio al que crece el consumo y tamaño no aumenta el perf. gain
- 5) - El software de hoy en día es altamente paralelizable (web server)
Hubo 2 approaches p aprovechar TLP
 - 1) Cmp Multi processing: 2 procesadores es 1 cmp (compartidos o no el code)
(poder tener muchos CMP en una config multiprocessor)
Pero el CMP es grande y ~~ff~~ de manufacturer
(y me hablar de power considerations) q' caso y connect ++
 - 2) Permitir que en solo procesador vaya alternando ^{threads} ~~threads~~
Switch on event o Time-Slice (tiempo fijo x thread)
no llegan a buena performance por no se llega
a una utilización óptima del DFB...

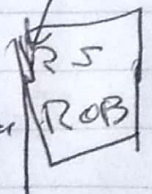
Seguimiento: Origen 2 Final: SMT



- Primera aparición en el **Xeon**, hay 2 next IP's
- 1 TC (8-way, Set Associative, LRU), entries tagged w/ Thread Info
- 2 ITLB
- 1 Allocator se encarga de colocar las plops en los buffers haciendo que **ningún thread escape + de la mitad de los entornos**
- 2 RATs

- 2 **Modes of operation**: ST or multi-Task
 In ST (0 or 1) se le dan todo el resto de los recursos (las mitades) a algunos de los dos procesadores lógicos. Un procesador se puede HALTear

El Xeon implementa la IA-32 y es buena p/servers



Elementos clave de NetBurst: BTC, OoO Engine & Rapid Exec Engine
Store-to-load forwarding (nos)

P4: The Microarch of Intel Pentium 4 processor on 90 nm Tech

- 1) Le hace mejoras al P4 original:
caches más grande, buffers más grande y
HyperThreading Tech
- 2) Esto basado en la microarquitectura NetBurst
~~Dynamic~~ **Static**
- 3) El ~~Static~~ Branch Prediction cuando no estaba lo dir en la BTB, (Branches Taken, predicted not-taken)
ahora también tiene en cuenta la **distancia**
- 4) Se añade un **indirect Branch predictor**
- 5) Xer a, a ya no genera dependencias

Execution Core:

- 6) - se añade un **shift/rotator block**
- 7) - Int multiply ya no se hace parando por floats
- 8) - 4+ reg (L1 cache Data)
- 9) + **buffers (schedulers)** + grande + **window for parallel**
= store-load predictor (?)

Mem Subsys

- 10) - L2 \rightarrow 1MB, write back, 8-way, 128 byte line
- 11) - Reduce cost of software prefetch on DTLB
- 12) - hardware prefetch
= 24 \rightarrow 32 withstanding store

Traces \rightarrow seq of pOps

Hyper Threading

- La mayoría de los cambios hacen sentido p/multiple threads. Ej. Outstanding loads that missed $4 \rightarrow 8$
- Una page table walk ahora puede suceder al mismo tiempo que un acceso a memoria que divide una línea de cache
- A la vez una walk que afecta todos los caches no impide que haya habiendo walks
- Context Identifiers (se agrega 1 bit a la cache) permite compartir mejor el L1 Data se anula threads usan la misma parte de la mem
- Se agregan **tracelp2**

M651 -- fast ++ cores
-- pipeline

⑤ Introduction to Intel Core Duo Processor with

- 1) - First Mobile micro that uses CMP (muchos núcleos en el chip)
- 2) - Basado en el Pentium M, mayor diferencia: dual core tiene que ver con que es muy costoso tratar de seguir dándole + eficiencia a ST y e mejor hacer + paralelismo (los diseñadores se dan cuenta y bajan la frecuencia 31 → 19 pipeline ↓ clock)
- 3) - 2 cores pentium M con shared L2 cache (ahora c/ virtualization, streaming SIMD SSE3, y más)
- 4) - Había un problema SSE a 128 bit-wide pero Pentium M core a 64 (ensancharla daba mucho calor)
y -- batería
había un problema en el decodé que se arregla c/ Op fusion y cancellation
- 5) - Se arregla un bottleneck de FP
- 6) - Se hizo + rápido el IDIV
- 7) - Se agregó H/W prefetching

CMP-STRUCTURE

- Cada core tiene su propio APIC
- Compatible con P4 que tienen HT tech!

Sleep state regla: - energía + time to wake up

POWER CTRL: 2 técnicas, se usa **SpeedStep** en

- 1) * Enhanced **Sleep States**
- 2) **Dynamic Intel Smart Cache** usage

Se puede manejar el sleep state de cada core por separado

Thermal Design Point

- Ahora (antes es Pentium M) hay muchos hot spots
se usa un **digital thermometer** (antes un diodo)
se puede consultar por software y event based reporting
- Power Monitor func: heat? → throttle CPU

Platform Power Management ... }

Intel Core SOLO PROCESSOR

- se desactiva un core p/ambiente de limited thermal ^{control}

5.3 CMP implementation in Systems Based on Intel Core Duo Processor, **MESI**

- Por qué CMP para mobile?
porque **dual core requiere menos potencia que incrementos**
la frecuencia

$$P = \underset{\substack{\text{potencia} \\ \downarrow}}{\propto} \underset{\substack{\text{capacitancia} \\ \uparrow}}{C} \underset{\substack{\text{voltaje} \\ \uparrow}}{V^2} \underset{\substack{\text{frec} \\ \uparrow}}{F}$$

por 101 MESI y SnooP }

- Se pensó en proteger o compartir el L2
En caso de Shareable: MESI o Directory?

Split cache? NO → bajaría el ST performance (only half cache & each core)
Tampoco permitiría share data en MT app

Se eligió MESI en vez de DCS pues es menos complicada y hubiera afectado primer corruption

THE PROTOCOL.

- Se mejoró la comunicación entre los cores
ej: el RFO puede ser muy rápido si es solo interno

Tiene el mismo MESI que en otros
Pentium 4

Penryn Processor Arch & Microarch

⑥ Original 4nm Intel Core 2 Processor Perf.

- Enhancements:

- 1) Larger L2 cache DualCore (6MB) Quad (12MB)
- 1) Faster divider
- 1) Shuffle Engine (128-bit wide)
- 1) Inclusion Filter
- 1) Renamed RSB (return stack buffer) ~~++ret pred acc.~~
- 1) CL1 & ST1 ++perf
- 1) Enhanced Intel Dynamic Acal Tech (improves energy efficiency)
- 1) Enhanced Intel Virtualization Tech
- 1) New SSE4 & BOOST Tech
- Conditional MOVES, Blend4, early out

++ performance para audio & video

- 1) Int Div is via CORDIC FP divide (Cointroller can Intel)

⑥.6 Improvements in the Intel Core 2 Penryn Processor Family Arch & Microarch

Improvements

- A) Super Shuffle (!)
se elimina la 64 bit unit c/ una unidad
que shufflea de a 128 bits a la vez
 - B) SSE4 (Ej RoundPS)
 - C) Streaming Reads (!)
Se pueden hacer lecturas muy rápidas de
regiones no cachables de memoria ~~es~~
(ej. graphics video card)
MOVNTQ
 - D) DPPS dot product (nive para
cellular detection)
 - E) New Radix-16 DIVIDER (4 quotient digits/iteration)
 - F) Int divs se hacen como float p/ ~~algoritmo~~
HW
- ↓ todo apunta a mejor performance de **MEDIA**
- G) CLI/STI performance ++